# Automatic Checklist Generation for the Assessment of UML Models

Tom Gelhausen, Mathias Landhäußer, and Sven J. Körner

Institute for Program Structures and Data Organization
University of Karlsruhe
76131 Karlsruhe, Germany
{gelhausen, lama, koerner}@ipd.uka.de

**Abstract.** Assessing numerous models from students in written exams or homework is an exhausting task. We present an approach for a fair and transparent assessment of the completeness of models according to a natural language domain description. The assessment is based on checklists automatically generated by the tool SUMO$\chi$ we present. SUMO$\chi$ directly works on an annotated version of the original exam text, so no 'gold standard' is needed.

## 1 Introduction

If we teach modeling we also need to grade models. But the task of modeling comprises a certain degree of freedom, and this renders exam questions on that task somewhat awkward: Assessments made by the examiners are frequently rejected by the examinees. This is either due to mistakes by the examiners or to unreasonableness of the examinees. No doubt, the examiners make mistakes as the task of assessing numerous models is exhausting, requires fast perception, and great mental flexibility. But as frequently, some students tend to challenge the adequacy of the assessments and beg to turn a blind eye here and there. We call this behavior 'salami slicing'. Experience shows, this tactic occurs for modeling tasks during homework reviews and post-exam reviews. However, *insight* on the students' side by itself should be worthwhile, no matter what the study regulations prescribe.

We identified a need for a procedure for evaluating modeling tasks that meets the following requirements: The assessments should be systematic, fair, transparent, and *cumulative* in a sense that they consist of simple atomic decisions which themselves are (ideally) unquestionable. Based on a technique which we initially designed to automatically generate domain models [1], we developed SUMO$\chi$ (SAL$_E$-based UML MOdel eXamination) to systematically create comprehensive and self-explanatory checklists that can be used to rate UML models[1]. The clou of our approach is that it does not use a 'gold standard', i. e. a model that would (after all) act as a yardstick for the students' solutions – and not

[1] Please note that the approach is not limited to class diagrams. We currently support class and sequence diagrams, state charts and OCL.

In the proceedings of the
Educators Symposium @ MODELS 2008
Part of the MODELS 2008 Conference, Toulouse, France
September 29, 2008, http://www.iem.pw.edu.pl/edusymp08/

only as a *sample* solution as it is usually declared. Instead, Sumo$\chi$ works on a semantic annotation of the original exam question's text and is therefore able to generate considerably less biased checklists than one would create by hand.

The checklists our approach generates are designed to measure the *completeness of the content* of a model according to the given domain description. All established metrics on models measure quality aspects of models such as similarity and class relationship complexity (cf. Section 5). In this respect, general checklists on model quality and the domain-dependet checklists our approach generates complement each other.

We start with the explanation of the approach and how it can be used to generate checklists. We then evaluate the complete process using the checklists to assess students' homework and exams. The lessons we learned during this procedure are listed afterwards. The following section focuses on the related work and also serves as foundation for our idea since it shows the necessity of such a tool. The article closes with a conclusion.

## 2 The Approach

Assume you want to give your students a modeling task as it is depicted in Figure 1. You provide a text describing the application domain and you expect your students to model it concisely regarding this text.

**Exercise 17:** Model the following domain in UML. You may use OCL to specify constraints.

*The game of chess is played between two opponents. They alternately move their pieces on a square board called a chessboard. The player with the white pieces commences the game. A player has the move, after his opponent made his move…*

**Fig. 1.** Example modeling task.

We propose to use a systematic checklist to grade the results from the students. Table 1 shows an excerpt of such a checklist for the text above. This checklist was generated by Sumo$\chi$ and has already been filled out by an imaginary examiner (in script style). It contains one section per sentence of the original domain description. For example, row 1 introduces the first section of this checklist. Each section lists all elements that could be modeled. For the first section, *opponents* and the *Multiplicity 2* are exemplified (rows 2-6 and 7-10). The *opponents* in turn give an example for a linguistic structure, a constituent

in this case, that can be modeled in various ways: They could be modeled as a class (row 2) or as a role (-name) within a UML association (row 3). SUMOχ also proposes that *opponents* could be modeled as an instance of another class (row 4) – possibly meaningless in this case but obviously meaningful in the case of the element *player* below (row 30). However, our imaginary examiner did not find a class, a role, or an instance to capture the concept of *opponents*. Instead, he found a note from the student that the student unified the classes *Player* and *Opponent* (row 6).

**Table 1.** Example of an Assessment Based on a Generated Questionnaire.

| Row | Element of Phrase | Modeled... | yes | no | wrong |
|---|---|---|---|---|---|
| 1 | **Phrase: The game of chess is played between two opponents.** | | | | |
| 2 | "opponents" | as class | | ✗ | |
| 3 | | as role | | ✗ | |
| 4 | | as instance | | ✗ | |
| 5 | Differently (please specify): ............ | | | | |
| 6 | Explicitly not modeled (please specify): Opponent- and Player-classes are equivalent .... | | ✗ | | |
| 7 | Multiplicity "2" of "opponents" | as multiplicity | | ✗ | |
| 8 | | in an OCL constraint | | | ✗ |
| 9 | Differently (please specify): ............ | | | | |
| 10 | Explicitly not modeled (please specify): ............ | | | | |
| ... | etc. | | | | |
| 27 | **Phrase: The player with the white pieces commences the game.** | | | | |
| 28 | "player" | as class | ✗ | | |
| 29 | | as role | | ✗ | |
| 30 | | as instance | ✗ | | |
| 31 | Differently (please specify): ............ | | | | |
| 32 | Explicitly not modeled (please specify): ............ | | | | |
| 33 | Attribute "white" of "pieces" | as Boolean function (with a parameter) | ✗ | | |
| 34 | | as scalar function | | ✗ | |
| 35 | | as state of "pieces" | | ✗ | |
| 36 | | as attribute of "pieces" | ✗ | | |
| 37 | Differently (please specify): ............ | | | | |
| 38 | Explicitly not modeled (please specify): ............ | | | | |
| ... | etc. | | | | |

Obviously, our approach maps linguistic structures[2] to model elements. This approach originated in Abbott's 1983 article "Program Design by Informal English Descriptions" [2]. Our method incorporates various suggestions that have been published in the domain of (automatic) model extraction since then (see [3] for a survey of approaches). Basically, we collected all UML fragments we found that were generated from single (simple or complex) linguistic structures; Table 2 enumerates some of the resulting mappings. Applying these mappings and generating the appropriate questions can be done by hand, of course. But up to

[2] On a syntactic level such structures are expressed via noun phrases or adverbials, for instance. But actually, we do not map *syntactic* patterns on the text source, but rather *semantic patterns* on the internal discourse model (cf. Table 2).

**Table 2.** Linguistic Structures matched to UML (excerpt).

| Linguistic Structure | Explanation | UML model element |
|---|---|---|
| object with the role **AG** | An acting person or thing executing an action | Class, role, or instance |
| object with the role **PAT** | Person or thing affected by an action or on which an action is being performed | Class, role, or instance |
| object with the role **ACT** (+AG +PAT) | An action, executed by AG on PAT, or a relation between AG and PAT | Method named ACT at the AG class with PAT param, association named ACT between AG and PAT classes, state or transition named ACT, etc. |
| $ACT_1$ OMN PARS $ACT_2$ PARS $ACT_3$ | A complex action (OMN) composed of multiple other actions ($n\times$PARS). | Subcall of $ACT_2$ and $ACT_3$ from $ACT_1$ in sequence diagram or in a comment on method $ACT_1$ |
| | *etc.* | |

now, we identified a total of 35 rules[3] to turn linguistic structures into questions about model elements. We created Sumo$\chi$ to apply these rules automatically and published an online version of the tool together with the ruleset [4].

### 2.1 Preparing the text

To use Sumo$\chi$, one first annotates the domain description using Sal$_E$. This is an annotation language for the markup of the semantics of natural language [1]. This task does not require any modeling knowledge, since it is just a linguistic annotation. Neither does it require any deeper grammatical knowledge (even though this might help), as the annotation concerns semantics, not syntax. The result of your annotation for the above example is shown in Listing 1. The original text is denoted in bold font whereas all added information is non-bold.

Roughly speaking, Sal$_E$ distinguishes objects, $n$-ary relations, and the roles that the objects play in these relations. As you can see in Listing 1, all clauses and all sub-clauses are embraced using square brackets (`[]`). Each pair of brackets denotes a relation. Within a relation, Sal$_E$ treats single words and nested relations as units. A word is commented out if it is prefixed with a hash (`#`). An object is marked by denoting its role(s) using a vertical bar (`|`) in conjunction with the role name(s). Examples for roles used in this excerpt are AG, ACT, and PAT. These three roles are also described in Table 2. (For a complete list of the 44 roles our system currently supports, see [1] or [5].)

[3] The exact number significantly depends on the 'intelligence' of the rule processor. The number 35 would apply to a human processor. A computer program is less fault-tolerant according to the applicability of the single rules and thus requires (in our case) a total of 120 rules for all variations.

**Listing 1.** $\text{SAL}_E$-Annotated Domain Description

```
1  [ #The game_of_chess|PAT #is played|ACT #between *two opponents|AG ].
2
3  [ They|AG move|ACT their|POSS pieces|{HAB,PAT} $alternately<<3 #on
4    [ #a $square ^board|{PAT,FIN} called|ACT #a chessboard|FIC ]|LOC_POS ].
5  [ @They|EQD @opponents|EQK ].
6  [ @their|EQD @opponents|EQK ].
7
8  [ [ #The ^player|POSS #with #the $white pieces|HAB ]|AG
9    commences|ACT #the game|PAT ].
10 [ @game|EQD @game_of_chess|EQK ].
```

In $\text{SAL}_E$, a nested relation (like in line four: *a square board called a chessboard*) can per se take a role in its outer relation. Yet, if an inner object of the nested relation is marked with a caret (^), it is interpreted as the head of the clause (like in ANTLR grammars). This head is then bound to the outer relation with the role which is appended to the nested relation. Thus the *board* in line four takes the roles PAT and FIN in the inner relation and the role LOC_POS in the outer relation.

Each object can be attributed in $\text{SAL}_E$. An attribute is denoted with a dollar sign ($) and applies to the following object – except when it is explicitly reassigned using the movement operators (<< and >>): The *alternately* in line three is reassigned to modify the *move*-action this way. A special form of an attribute is the multiplicity which is denoted using an asterisk (*).

As $\text{SAL}_E$ is rather a compiler than a sophisticated natural language processor (cf. Section 2.2), we need a simple and deterministic way to cope with coreferences. There are two ways in $\text{SAL}_E$ to express coreferences: either via word occurrence references or via assertions. A word occurrence reference is denoted with an at-sign (@) in front of a word. It tells the system, that a word refers to the very same thing (same instance) as in the preceding occurrence of that word – not just to "any instance of this category", which is the default. The lines five, six, and ten contain assertions, expressed via relations with special roles. These relations do not directly originate from the text. But the annotator has entered them to tell the system that, for instance, *they* and *opponents* means the same thing. The role EQK denotes the one of the two *eq*ual concepts that should be *k*ept in the internal discourse model, while EQD denotes the one that should be *d*ropped. The object that has been annotated with EQK inherits all roles and attributions from the dropped object in this case. For a more detailed description of this process see [5].

## 2.2 Implementation

The questions are generated based on pattern matching on the discourse model. Therefore, $\text{SUMO}\chi$ uses the $\text{SAL}_E$ compiler to obtain this discourse model from the annotated text. Technically, the $\text{SAL}_E$ compiler is based on an ANTLR [6] generated parser, thus avoiding resource consuming and error-prone Natural

Language Processing (NLP). It builds a graph representation of the discourse for the graph rewrite system GrGen.NET [7]. The graph rewrite system then executes the above-mentioned 35 (respectively 120) declarative transformation rules. Finally, the surface structures (i. e. the actual questionnaires) are generated via the application of XSL-T templates.

An optimization that we implemented in the rule set for GrGen.NET concerns the repetition and the order of the generated questions: The rules do not produce more than one question about elements that are mentioned in the domain description more than once. Because of this, the list contains questions about the existence of elements (for example classes) in the beginning, and questions about their behavior (methods, state transitions) or their design (attributes, associations) later on. This eases the assessment since the examiner first needs to find the element before he can answer questions about it. However, this process to generate checklists is systematic and straightforward. Some manual rework of the resulting lists is necessary, but in our experience, this is largely limited to deletion of superfluous questions and adaption of flections. We regard this as a negligible issue as a user of SUMO$\chi$ will probably revise the list for his own version, for example including model quality criteria.

## 3 Evaluation

We evaluated our approach in various ways. First of all, we successfully used it for grading exams and homework – and plan to do so in the future. For exams, we determined student complaint rates. We compared these rates with historical data on non-modeling questions. The results suggest that assessments created this way are acceptable to the students. Yet, as the historical data on exam results has not been collected with this research in mind, we currently cannot conduct other studies on exam results and the students' complaint behavior. To further evaluate our approach, we conducted an extensive study on a large homework in which we asked pairs of students to model the game of chess. We provide statistical analyses of the results which show a high inter-rater agreement. Besides the numerical results, we examined the discrepancies of the examiners' ratings on a qualitative basis. We present our insights in Section 4.

### 3.1 Legitimate Complaints in Exams: 'Intra-student' Fairness

Written exams for our software engineering lecture are held twice a year and take 60 minutes. Typically 60 to 250 graduate students of computer science attend the lecture and go for an exam. After the assessment, we hold a post-exam review in which the students can double-check. If they find grading errors, they can ask for correction. After this review, the final grade is determined. Approximately 45 % of the examinees attend these reviews.

Two recent exams included a modeling task worth 17 of 60 points. We advised our students to examine the text carefully and to closely analyze every part of every phrase on its own. Since they were pinched for time, we did not expect their

models to be overly sophisticated, but that they kept closely with the domain description.

We compared the results of these two modeling questions with 22 questions of other types from written exams over the last three years. These 22 questions are of well established types, for example memorization questions such as ‘Name three stages of the waterfall model’. These types can be assessed in a fair and transparent manner and lead to few legitimate complaints in the post-exam reviews. In the two modeling tasks, the students achieved moderate results, so complaints were to be expected. During the post-exam reviews, we handed out the filled-out checklists along with the exams. This way, the ratings were completely transparent to the students. A low number of legitimate complaints would be considered fair for each individual student[4].

To compare the assessments of the questions, we computed the post-exam review gain $q_e$ as the mean difference between the results before and after ($p^{pre}$ and $p^{post}$) the post-exam review for every exercise $e$ for all students $s$ that reviewed their exam (let $n$ be the number of these students). The post-exam review gain was normalized by the maximum number of points $p_e^{max}$, that could be reached in the corresponding exercise: $q_e := \frac{1}{n} \sum_{s=1}^{n} \frac{p_{es}^{post} - p_{es}^{pre}}{p_e^{max}}$. The results are depicted in Figure 2: The post-exam review gain of the 22 non-modeling questions (left side) are opposed to the two modeling tasks (right side) in this diagram. Every triangle and both squares represent the post-exam review gain for one exam question. The average $n$ (i. e. the number of students that reviewed their exam in one of the six review sessions under scope) is 64.8. As can be seen, the post-exam review gain of the modeling tasks is pleasingly low. Obviously, our way of assessing UML models can keep up with other (evolved) types of exam questions.

### 3.2 Homework Study: ‘Inter-student’ Fairness

Our experience from the post-exam reviews shows that assessments are repeatedly biased by the examiners’ different levels of generosity (in the sense of their margin of discretion). Nonetheless, an assessment method should still be monotonic in a sense that better students always get better grades. We conducted a study where multiple examiners independently assessed a randomly chosen set of student homework. A high inter-rater agreement would indicate that the students rank independently from the examiners. This study was run to verify this property in order to ensure inter-student fairness.

We chose the FIDE laws of chess as domain description to be modeled in UML. This resulted in a domain description of four pages. The students attended the software engineering lecture and were mainly graduate students of computer science. They were allowed to work as two-person teams and had to develop a UML model accompanied with OCL as part of an assessed homework. For this study, we randomly picked four student pairs and asked four examiners from the

[4] We suspect that in this case every student feels *his* assessment to be profound. We examine ‘inter-student’ fairness in Section 3.2.

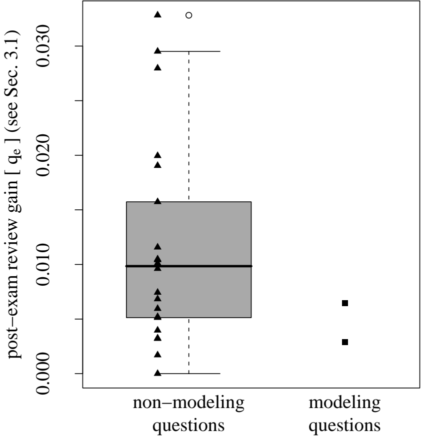


**Fig. 2.** Legitimate complaints about assessments (post-exam reviews).

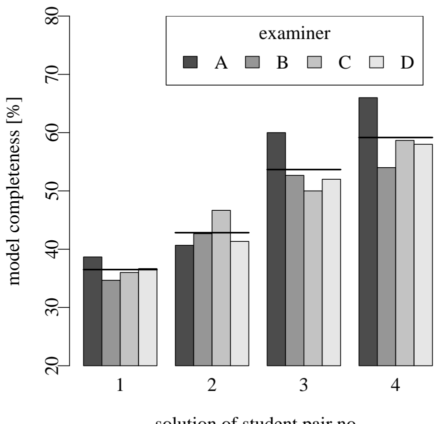


**Fig. 3.** Comparison of ratings (assessed homework).

staff of our chair to rate their work. The diagram in Figure 3 shows the evaluated model completeness of the inspected UML models. The horizontal line indicates the examiner groups' average rating for each model.

In order to compare the agreement of every possible examiner pair, we computed Pearson's correlation coefficient[5] $r$ (for details see [8]) of each examiner with every other. Since we expect the results of the examiners to be positively correlated, we try to reject the null hypothesis $H_0 : r \leq 0$. We assume the null hypothesis to be rejected if the $p$ value calculated with a one-tailed t-Test is below the significance level[6] of $\alpha = 0.05$.

**Table 3.** Pairwise correlation with Pearson's correlation coefficient.

| $(r,p)$ | examiner A | examiner B | examiner C | examiner D |
|---|---|---|---|---|
| examiner B | (0.904,4.8%) | – | – | – |
| examiner C | (0.826,8.7%) | (0.935,3.2%) | – | – |
| examiner D | (0.964,1.8%) | (0.968,1.6%) | (0.945,0.3%) | – |
| avg | (0.961,1.9%) | (0.976,1.2%) | (0.945,2.7%) | (0.999,0%) |

[5] The values of $r$ can range from $-1$ to $+1$ and indicate strength and direction of a linear relationship between two variables $X$ and $Y$, in our case of the scores for the models of two examiners. A value of 0 means that we have no linear relationship between the variables. Positive values indicate a positive linear relationships, i. e. whenever the value of $X$ increases, the value of $Y$ increases, too. Negative values indicate a negative linear relationships, i. e. whenever $X$ increases, $Y$ decreases.

[6] Meaning the probability that our observed correlation coefficients $r$ (or greater coefficients) could be due to chance is less than 5 %.

The first three rows of Table 3 show the correlations among the examiners. The values of $r$ suggest that there is a strong linear relationship between the results of the examiners. Only the $p$ value of the correlation between examiner A and C is greater than the significance level $\alpha$, but the probability that the corresponding correlation coefficient is due to chance is still below 9 %. Even though this value does not allow us to reject $H_0$ on a significance level of $\alpha$, we support the conclusion that our approach is indeed successful.

To investigate the consistency of the examiners' rating with the average group rating, we calculated Pearson's correlation coefficient of every examiner with the group average. The resulting correlation coefficients and their corresponding $p$ values can be found in the last row of Table 3. Again we found high values of $r$ suggesting a strong linear relationship between the ratings of the examiners and the group's average rating. This time, all $r$ values are statistically significant. This clearly states that the decision of a single examiner is consistent with the group's decision.

## 4 Observations and Discussion

As our homework study showed (cf. Figure 3), we have not reached absolute unquestionableness yet. Comparing the different examiners' results, we observed that they are still biased – so there must be room for interpretations. This does not impose a threat to assessments conducted by a single examiner (or a small group of examiners who interact and synchronize their decisions): In our study, the examiners were strictly isolated to ensure statistical independence. Yet, after the assessments, the examiners discussed the discrepancies in their decisions.

### 4.1 Observations Concerning the Examiners

Examiners had different interpretations of the models and the meaning of some list items. We need to minimize this margin of discretion. An example is the rook's multiplicity which could have been 2 or [0..2] or [0..10]. The latter is possible through pawn-promotion.

After a while, the examiners tended to mark list items which were clearly non-existent in the students' models. This could be due to the fact that only having parts of the model sometimes resulted the examiners forming their own – more complete – model in their mind. For example some examiners marked classes as modeled though they were only *indicated* in a state chart.

A huge effort is finding the sought-after item, especially if the models comprise multiple pages (as in the case of our homework study). UML diagrams in digital form could ease this effort as computer-based find functions could speed-up the process. Exams might still be written by hand, but don't usually exceed a certain model size due to time constraints.

It is also hard to ensure that examiners find and check all possible versions: If an element occurs multiple times in the model, it also has to be marked multiple times in the list in our study. But the examiners tended to stop searching after an element was found for the first time.

### 4.2 Observations Concerning the Students

Some students delivered models that were hardly UML. Collectively using a UML tool should help to check the models for compliance with the rules of UML, thus improving the proof-reading-process. UML tool use is impossible during exams.

A number of students handed in code rather than models. This contradicts the idea of abstraction and rather gives a concrete implementation including a fixation on a certain kind of technology. This misunderstanding can be prevented by clear formulation of exam questions.

Some students used domain knowledge to complement their models: They added artifacts which were not given in the domain description. In turn, some students left out parts that definitely were in the description, for example the (rather complex) castling move. We conclude that some of the students did not focus on the provided text, but rather modeled the game by what they knew by heart. Our approach of making content-completeness the primary basis of valuation leads to bad grades for these students. The examiner must decide whether this poses a threat to the validity of his ratings.

### 4.3 Applicability

The process we presented here still depends on the provided text to a large extend. In comparison to many other model extraction approaches (see [3], for example) $\text{SAL}_E$ is much less dependent on the chosen formulation. Nevertheless, our approach can only work with content that a) actually resides in the provided text and that b) is also annotated correctly.

The effort for the annotation obviously depends on the experience of the annotator, but it also depends on the given domain description. This effort may also tip the scales regarding the applicability of our approach in industry. Indeed, there seems to be a need for a metric for content-completeness of models, as we show in the next section. On the one hand, the effort in learning $\text{SAL}_E$ mainly comes from learning and applying the 44 roles we currently support. Practice will show whether and how this role set can be optimized in the future. On the other hand, effort originates from 'linguistic defects' in the domain description. Linguistic defects are unobvious semantic or pragmatic flaws in the text, such as deletions, generalizations, and distortions (see [9] for details). Experience shows that (roughly spoken) better specifications are easier to annotate.

Concerning the grading of models, the approach is limited in two ways. First, it is not suited to grade (object-oriented) analyses, it has been designed to grade models of a certain, precisely prescribed domain. Second, it is not suited to assess architectural decisions that might be contained in a model. Both limitations are by design – even though we support the opinion, that analysis and architecture are two important subtasks of modeling in practice.

Regarding the first limitation (grading analyses), we understand that this is a challenging but particularly a different task. Though technically, it could be done with our approach by assigning the task to the students withholding certain aspects (i. e. *the expected*) from the domain description. The philosophical

question behind is how fair such expectations can be: The examinee not only has to catch (guess) all of the expected aspects, he or she also has to guess the intended degree of relevance for the model. We have no idea, whether (and how) the omitted aspects can be made "obvious enough". Concerning the second limitation (assessing architectural decisions), there are already plenty of metrics available. These can be used to complement our approach without difficulties.

## 5 Related Work

Grading models is exhausting as there are usually countless correct solutions: Lange et al. [10] show that UML models bring the freedom of creativity to the modeler because they can be ambiguous *and* correct at the same time. The lack of uniformity and the large amount of defects contained in UML models result in miscommunication among UML readers. Lange et al. [10] propose modeling conventions, analogous to coding conventions for programming. Results indicate that decreased defect density is attainable at the cost of increased effort when using modeling conventions. This shows that assessing UML models remains a complicated – and not yet mastered – task. Modeling conventions could probably be enforced by tailoring our assessment questions to the proper settings.

Lange and Chaudron show that modelers who tend to exploit the freedom of UML impose risks on software projects [11] and their modeling phase [12]. One has to take into account that even unambiguous and syntactically correct models can be incomplete with regard to the users' needs and specification [13].

Mohagheghi and Aagedal [14] agree with our basic assumption that model content-completeness is important. They ran a general survey of various methods on determining model quality. As it turns out, quality assurance techniques must not only consider the quality itself but also the completeness of UML models. The earlier one focuses on model completeness, the better.

In 2006, Lange and Chaudron [15] conducted a survey among practitioners which analyzed UML models in 14 case studies. Even though 52 % of the practitioners responded that model completeness is a major stopping criteria for their modeling activities, there are hardly any metrics to measure completeness. Incompleteness is a big problem: More than half of the reported miscommunication between project stakeholders in subsequent software process stages is due to incomplete models. Stakeholders would like to use metrics two to four times more often than they actually do. Briand et al. [16] point out that working with complete models progresses faster and delivers higher quality software. There clearly is a need for assessing the completeness of models – even in industry.

Being able to judge the quality of a UML model is important to software quality. No known approach considers checking the models against the domain description. But these models are eventually used to create the software. Being the prime artifacts of Model Driven Engineering, the models are highly important for the quality of the resulting product [14]. But how can a model be evaluated?

There are some software metrics which are applicable to UML models [17], but most of them are made for analyzing source code. These metrics are usually

computed in the last stages of the software process – stages in which a metric cannot show the deficiencies of the product; only the defects of a potentially incomplete implementation. Therefore the model's quality should be evaluated long before the resulting source code itself is created. But the fact that the models deviate from the initial (customer's) textual specification is not yet dealt with. McQuillan and Power show that we need to ensure the software follows the fundamental ideas of the customer [18]. Our approach makes matching the content of a textual specification to existing models feasible.

## 6 Conclusion

We presented Sumo$\chi$, an approach to automatically generate comprehensive and self-explanatory checklists for the assessment of models. These checklists are intended to grade the content-completeness of models according to a given domain description, but they could as well serve as a new metrics for the content-completeness of models. A great advantage of our approach is that it does not use a 'gold standard'. Instead, it creates checklists that do systematically *not* restrict the freedom of the modelers. We reached our major design goals of being systematic, cumulative, transparent, and fair. Yet, our studies showed, that there is still room for interpretation while working through the checklists. This leaves room for improvement in future work. Besides an optimization of the role set, tool support for $\text{Sal}_E$ could ease the annotation process.

## References


1. Gelhausen, T., Tichy, W.F.: Thematic Role based Generation of UML Models from Real World Requirements. In: First IEEE International Conference on Semantic Computing (ICSC 2007). Volume 0., Irvine, CA, USA, IEEE Computer Society (September 2007) 282–289
2. Abbott, R.J.: Program Design by Informal English Descriptions. Communnications of the ACM **26**(11) (November 1983) 882–894
3. Moreno, A.M.: Object Oriented Analysis from Textual Specification. In: Ninth International Conference on Software Engineering and Knowledge Engineering, Madrid, Spain (June 1997)
4. Landhäußer, M.: Sumo$\chi$. Available online: `http://www.ipd.uka.de/~lama/sumox/` (August 2008)
5. Landhäußer, M.: Automatische Erzeugung von Prüflisten zur spezifikationsbezogenen Beurteilung der Vollständigkeit von UML-Modellen. Studienarbeit, Institute for Program Structures and Data Organization (IPD), University of Karlsruhe (July 2008)
6. Parr, T.: ANTLR. Available online: `http://www.antlr.org/`
7. Geiß, R., Batz, G.V., Grund, D., Hack, S., Szalkowski, A.M.: GrGen: A Fast SPO-Based Graph Rewriting Tool. In Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G., eds.: Graph Transformations - ICGT 2006. Lecture Notes in Computer Science, Springer (2006) 383 – 397 Natal, Brasil.
8. Howell, D.C.: Fundamental Statistics for the Behavioral Sciences. 4th edn. Brooks/Cole Publishing Company (1999)

9. Rupp, C.: Requirements and Psychology. IEEE SOFTWARE **19**(3) (May–June 2002) 16–18
10. Lange, C.F.J., Bois, B.D., Chaudron, M.R.V., Demeyer, S.: An Experimental Investigation of UML Modeling Conventions. In: MoDELS. (2006) 27–41
11. Lange, C.F.J., Chaudron, M.R.V.: Managing Model Quality in UML-Based Software Development. In Chaudron, M.R.V., ed.: Proc. 13th IEEE International Workshop on Software Technology and Engineering Practice. (2005) 7–16
12. Lange, C.F.J., Chaudron, M.R.V.: Effects of Defects in UML models: An Experimental Investigation. In: ICSE '06: Proceeding of the 28th international conference on Software engineering, New York, NY, USA, ACM (2006) 401–411
13. Lange, C.F.J., Chaudron, M.R.V.: An Empirical Assessment of Completedness in UML Designs. In: EASE '04: Proceedings of the 8th Internation Conference on Empirical Assessment in Software Engineering. Volume 920. (May 2004) 111–119
14. Mohagheghi, P., Aagedal, J.: Evaluating Quality in Model-Driven Engineering. In: MISE '07: Proceedings of the International Workshop on Modeling in Software Engineering, Washington, DC, USA, IEEE Computer Society (2007) 6
15. Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: In Practice: UML Software Architecture and Design Description. IEEE Software **23**(2) (2006) 40–46
16. Briand, L.C., Hove, S.E., Labiche, Y., Arisholm, E.: Guiding the Application of Design Patterns Based on UML Models. Proceedings of IEEE International Conference on Software Maintenance (ICSM), Philadelphia, USA (2006) 234–243
17. Kim, H., Boldyreff, C.: Developing Software Metrics Applicable to UML Models. In: Proceedings of QAOOSE'2002, QAOOSE 2002 (2002)
18. McQuillan, J.A., Power, J.F.: On the Application of Software Metrics to UML Models. In: MoDELS Workshops. (2006) 217–226